N91-20694

RULE-BASED SIMULATION MODELS

JOSEPH L. NIETEN
KATHLEEN M. SERAPHINE

LOCKHEED SPACE OPERATIONS
TITUSVILLE, FLORIDA

A B S T R A C T

This paper describes procedural modeling systems, rule-based modeling systems and a method for converting a procedural model to a rule-based model.

Simulation models are used to represent real-time engineering systems. A real-time system can be represented by a set of equations or functions connected so that they perform in the same manner as the actual system. When the real-time system is being modeled, the modeler will have to code the system's calculations and characteristics using some computer language or modeling tool.

Most modeling system languages are based on FORTRAN or some other procedural language. Therefore, they must be enhanced with a "reaction" capability. This reactive capability allows the model to perform only those calculations which are dependent on information that has changed. Even with this capability, a procedural model must look at every variable to determine if a calculation which depends on that variable should be performed.

Rule-based systems are reactive by definition. Once the engineering system has been decomposed into a set of calculations using only basic algebraic unary operations, a knowledge network of calculations and functions can be constructed. With the network in place, the rule-based system merely reacts to changes in the data. When a variable is changed, those calculations which depend on it become active. The rule-based system will continually execute, performing all dependent calculations appearing on the agenda.

The knowledge network required by a rule-based system can be generated by a knowledge acquisition tool or a source level compiler. The source level compiler would take an existing model source file, a syntax template, and a symbol table and generate the knowledge network. Thus, existing procedural models can be translated and executed by a rule-based system.

Simulation

Simulation is used in the real world to represent real systems, so that users can perform operations on a system that reacts the same as the real system without the cost or danger involved with the real system. Simulation systems allow an engineer to perform 'what-if' scenarios against their designs of real systems. Dangerous situations, expensive failures, and real life incidents can be recreated without risk using simulation. The newest use of simulation systems is in the training community. The training world has been incorporating simulation models into their training systems. The difference between a simulation system and a training system is an intelligent tutor in the training system, which interacts with the student and the simulation models to provide a high fidelity training session.

Simulation Models resemble mathematical models in that both have the same purpose and both utilize mathematical relationships to

represent real systems. However, closed-form, analytic equations do not represent complex real systems without some enhancements. Simulation models use the black box approach. Each component of the real system is represented by a 'black box' with inputs and outputs. Further, each box is a complex combination of continuous and discrete functionality, which must be represented by some mathematical relationship while preserving the continuous nature of some of the calculations. This is accomplished by maintaining the current state of each variable used within the model, while performing the numerical analysis dictated by the functionality of the 'black box'. A great deal of temporal reasoning is required of the modeler during this decomposition process.

Real-time execution of simulation models is an important concept/issue. In order for a simulation model to be considered real-time, it must be at the exact same state as the real system at any given point in time. This means that both the computer hardware and the model executive must be capable of performing a tremendous number of calculations per second. Further, the model executive must minimize the number of calculations required, so that real-time performance becomes a direct factor of the computer hardware. This means the model executive must be reactive and only schedule those calculations which have been stimulated. A calculation is stimulated when a variable which provides input to that calculation is changed. Not every application requires a real-time model, some can function with a near-real-time model. This is a model which is in sync with the real system some acceptable percent of the time, or has no more than a certain delay when providing output.

Procedural Simulation Methods

Simulation models have been developed using a variety of generic languages such as FORTRAN, COBOL, PL/1, C, and Pascal. There are even models which have been developed using tools which are based on one of these languages. Those model based on the newer tools have the ability to represent continuous and discrete systems, and can schedule calculations to some degree. However, in general, artificial intelligence features in the standard commercial simulation languages is not yet available, mostly they still rely on a procedural approach to performing simulation operations.

A procedural model will execute in a linear manner with well defined entry and exit points. This means, that the model must perform all operations that are between the entry and exit points of a particular procedure. Normally, this would not have an impact, however, the simulation process may require other areas of the model to perform calculations based on the new information generated by the calculation that just finished. In addition, that calculation itself may actually have to be performed again. Now, if the computer is busy finishing one set of calculations, it cannot (not yet anyway) also begin to perform any other calculations in tandem. Even if the computer could perform these tasks, how is it going to know that these tasks need to be performed - unless they are scheduled.

There are some systems today which can schedule tasks or calculations for execution. These systems take a procedural model and segment it into smaller pieces, which can then each be executed from start to finish without any need to communicate with the other segments. These segments can also be managed by an executive which schedules them for execution based on some ranking algorithm. This is a highly efficient method for performing complex simulations, however, this modeling system still has performance considerations. Even so, there are times when the amount of data which is changing overwhelms the model executive and it falls behind, thus making it less than real-time. This is a common problem when the real system is very complex and/or very large.

While segmenting a procedural model is very effective, the

segmentation does not go down to the lowest level of execution. By segmenting a procedural model into some number of calculations, the size of the segment can be limited. However, there are still some calculations within that segment which do not have to be performed. The only way to absolutely perform only those calculations which have been stimulated is to segment down to the lowest possible level. Once this segmentation is accomplished the final representation can be analyzed and optimized. This level of analysis and execution is going to be very difficult using a procedural methodology or tool.

Neural Simulation Methods

Neural Simulation Modeling represents an expansion of Neural Network and Rule-Based Programming techniques. Neural Simulation Models have two main components: A rule-based model executive and a neural network.

The model executive contains explicit rules which define how to interpret and perform operations on the information contained within the neural network. A basic knowledge of unary mathematic operations, as well as a working knowledge of the more complex functions which are used to represent physical systems mathematically, are the heart of the model executive. All function operations and variable information are imbedded within the neural network. So, the executive must take the function and variable information stored in the associated nodes and adjust the contents of the current node depending on that information. When a node is changed, the model executive traverses the network, performing any operations which are dependent on the updated node. By using a rule-based executive, this process can be totally reactive, which implies a higher degree of efficiency (in general) than procedural execution of models, since nodes will only be adjusted as they are stimulated. Thus, a stimulus driven environment is required for execution of the model executive.

Even though a stimulus driven environment is ideal for model execution, some controls must be added. Some calculations are self stimulating; that is they update a variable which also provides input for the calculation. This can trigger an infinite loop, since most rule-based systems rely on recency to schedule a rule for execution. This problem can be controlled by forcing those types of calculations to cycle in a two stroke fashion, similar to a two cycle lawn mower engine. In other words, those calculations which can trigger their own execution will not be able to do so until the next cycle.

The most difficult problem with implementing a rule-based model executive is timing and/or calculations dealing with a change over time. The impact of this problem can be avoided if some simple steps are taken. Those calculations which require a 'Delta T' must be time stamped. Whenever the calculation is executed, the time that the last execution occurred must be available to calculate the time elapsed. Since every system has some kind of internal clock or time elapsed register, this impact becomes an issue of machine cycles required to do the time calculations versus performance. Another possible impact occurs when a model tries to use a time delay for calculations. This impact appears to effect only those operations which are binary in nature. Therefore, this impact can be minimized with better modeling techniques.

The neural network uses nodes to store data and function information and uses connections to define input, output and directional characteristics. This type of representation allows for the most powerful use of the knowledge stored in its nodes. The mathematical equations which represent the modeled physical system have a sophisticated connection scheme, when they are all combined for the purposes of simulation activities. The most efficient way to represent some physical system would be to first identify the independent variables within the system. Then identify each level of abstraction which

relies on those independent variables, and continue this process until all levels of abstraction have been identified. The final result should be those variables which are considered 'output only'.

As a result of this process, all nodes in the network will be unique. This eliminates redundant calculations and operations. Once a network representation is in place, the network can be 'scanned' for closed operations. These are operations which can be calculated in a linear or procedural manner without impacting the efficiency of the network. In actuality, these operations can be replaced by another abstract function, which is in turn added to the model executive. This is not a trivial task, since the system may try to over simplify the network representation and inadvertently change the functional representation of the network. Therefore, a degree of intelligence is required when performing any network operations/optimizations.

CLIPS was chosen to be the platform for this development effort. CLIPS is versatile and portable; making the decision relatively easy. It is also a NASA product, so the product life was not an issue. CLIPS is not the perfect solution, at least not version 4.3. The activation algorithm could be optimized to improve real-time performance and there are not any intrinsic timing functions, which are required to do some real-time calculations against time. Even so, CLIPS is the best software for the job at hand.

CLIPS has a high potential in the distributed processing arena. Tasks which are queued on the agenda could be executed on any available processor, thus, increasing real-time performance. This would enable simulation performance and capabilities to be directly related to the hardware platform.

Neural Model's can be built using a Knowledge Acquisition tool or some other Graphical User Interface. This is a major improvement in the simulation arena. This also represents the ultimate situation, where older models can be converted to a newer technology and then maintained using a state of the art GUI. Another scenario could be that once the model is converted, you do not have to use a GUI. Instead, you could continue to build models with the old system and use the new compiler to debug or test the model before it is put into production on the simulation system. This alone could have a major impact on productivity.

Conversion Techniques for Simulation Models

Why would anyone want to convert simulation models? This is the one of the most frequently asked questions in the simulation community. Simulation systems are very complex by nature. They can also be dependent on some specific computer hardware for execution. As a result, these simulation systems are not portable to other computer hardware platforms. The technology base line has changed considerably in the last five years making the computer hardware developed during that time 'obsolete'. Hardware obsolescence is forcing the migration of simulation systems to more modern platforms. The user community has identified the need to consider portability when making decisions concerning the future of their simulation systems.

The decision one must make during model conversion is whether to convert the models themselves to a new and improved modeling system or to convert the compilers and executives used to generate and/or execute these models. Something to consider when making this decision is the composition of the job. The actual 'coding' of the model accounts for only 50% of the overall task. The remaining 50% of the work is spent doing the mathematical analysis of the physical system. In other words, at least 50% of the development time is spent building the knowledge about the physical system being modeled, into a mathematical system. The number of man-hours invested in this component

of development can represent a large investment. Therefore, it makes sense to convert these models at the source code level in order to retain the investment made in building the mathematical representation of the physical system.

Source level conversion can be complex. However, the real trick to being able to do a conversion of this type is the method of building the new mathematical representations from the old formats. While developing a new compiler is not a trivial task, it is a lot cheaper than the other options and certainly better than rewriting several hundred thousand lines of code.

The source level compiler takes an existing model source file, a syntax template, and a symbol table and generates the neural network, which is used by the model executive to perform the simulation. The compiler uses a mapping function to transform the original syntax into a modified neural network. The network must then be optimized in order to be used efficiently. It must have all duplicate nodes or covers (groups of nodes) combined and/or eliminated. All nodes not explicitly listed as either inputs or outputs must be eliminated as well.

The model compiler has to have a degree of intelligence, so that it can identify when to stop performing optimizations. It also should have the capability to request clarification on a node's status. The compiler should also be able to isolate work-around techniques used by the programmer. Periodically, programmers will develop a method for accomplishing some high order function, which was not available through the standard functional syntax of the original modeling system. This 'work-around' should appear in the network as a pattern where it can be identified and dealt with. A work-around can either be transformed or deleted depending on the mapping function.

Another method of conversion would involve porting the development tools to a new platform. This will allow for the continuation of existing model source. However, translating the base language of the development tools into a new language will not necessarily improve any of the internal algorithms nor will it guarantee portability. Portability will always be a major issue. In order for any product to survive in today's rapidly advancing technical world, It must be portable. Productivity is also an issue, since the development tools may be outdated to begin with. Converting outdated tools would be equivalent to giving an older car a paint job, but not a new engine.

Conclusions

Neural (Rule-based) Simulation techniques are extremely powerful and even though there are some problems with the implementation of this technology, neural models can provide the high capacity data manipulation required by the most complex real-time models. This technology is worth further investigation.

References

1. J. W. Schmidt, "Introduction to Simulation,"Proceedings of the 1984 Winter Simulation Conference, Dallas, TX, Society for Computer Simulation, San Diego, CA, 1984

2. R. Shannon, "Artificial Intelligence and Simulation," Proceed ings of the 1984 Winter Simulation Conference, Dallas,TX, Society for Computer Simulation, San Diego, CA, 1984

3. W. M. Holmes, Artificial Intelligence and Simulation, Society for Computer Simulation, San Diego, CA, 1985.

4. D. E. Rumelhart, G. E. Hinton, and J. L. McClelland," A General Framework for Parallel Distributed Processing," Parallel Distributed Processing: Explorations in the Microstructure of Cognition, VOL. I, MIT Press, Cambridge, MA, 1986

5. B. P. Ziegler, Theory of Modeling and Simulation, Wiley, New York, 1976